CHAPTER 13
I CAN SEE YOU

As far as USB is concerned, digital video is just like digital audio except there is a lot more data! Digital video uses the same isochronous features of the USB architecture, so there is no new USB theory to learn. The high data rate of digital video however causes the operating system to be much more involved in the transfer of data; the Windows operating system uses a kernel mode streaming driver to process all video operations and provides access and configurability via filter graphs (these are explained in detail later in this chapter). We shall see that inputting video into a PC host is a complex and specialist function and dedicated components are build for this task. Each manufacturer solves this function in a different way but all provide a Windows device driver to integrate their components into the Windows DirectShow architecture.

It is beyond the scope of this book to explain how these dedicated components are designed. My goal is to show how they can be **used** in a USB-based project. I will present three OEM solutions which are also purchasable as end customer products. I recommend that most readers buy one of the end customer products and adapt it to suit your application. I will discuss several software examples in this chapter to get you started. If you intend to build thousands of your products then I recommend that you work directly with the OEM supplier – they have the expertise you will need to make major modifications to the three hardware examples that I will cover.

My three hardware examples are all USB 1.1, or 12Mbps, examples. At the time of writing there were no USB 2.0, or 480Mbps, cameras available. I am eagerly awaiting the arrival of 480Mbps camera since the increase in bandwidth will mean higher quality, and more, images. Please check this book's companion web site – I will extend this chapter as soon as possible there.

DIGITAL VIDEO APPLICATION

In this chapter, the creation of a digital video application will involve choosing a chipset with the features you need and then writing and/or configuring software on the PC host. So this is mainly a software exercise and I will describe a variety of software tools that will simplify this task. You will see little "USB-ness" since this is hidden by the WDM device driver. I did consider going through the creation of an isochronous WDM device driver but it got terribly complex very early in the project and there was little design flexibility anyway. So I decided it better to **use** the built-in capabilities of the operating system and create something useful with the video from the USB camera. The USB connection does make this project run more smoothly since the class driver manages all of the low-level communications for us.

I will present three hardware designs. The first is a video conferencing camera, the second adds audio and the third adds audio and video out capability as well as audio and video in. All three come with their WDM filters that will be used in our software examples later in this chapter. Finally I present a selection of video-enabled applications to stimulate your imagination of what can be done with this USB video camera technology.

Before jumping into solution space I did want to cover the problem of inputting video on a 12Mbps USB connection. It is time to look at the amount of real data that we can put into a USB frame. Digital video will start to push the bandwidth limits of USB, so a good understanding of the actual packet timing is required to successfully implement a digital video I/O device.

We know that USB transmits packets of data at 12 Mbps in 1-ms frames. The following discussion will use the variable "Byte Times per Frame," or BTF, to identify how much data throughput can be expected. For a full theoretical analysis, see Chapter 13/USB Bandwidth Analysis on the CD-ROM. A summary of this paper is given here.

The maximum raw number of BTF is (12 Mbps / 1-ms) / 8bits/byte = 1500. We must reduce this number by the protocol overhead required to manage the link. Specific sources of overhead include packet organization, Start-Of-Frame and End-Of-Frame signaling, clock adjustment, and time reserved for control transfers. This reduces the useful BTF to about 1300.

Many of our early examples transferred small amounts of data between the PC host and the I/O device. Interrupt packets were used to guarantee delivery, so the transactions consisted of {Setup-Data INorOUT-Handshake} sequences. At full speed, a one-byte data packet is equivalent to 12 BFT, and at low speed this is (12*8) + (2 for preamble) = 98 BTF. Even at low speed this is a small load on

USB, and in fact the actual load is much smaller than this because these interrupt packets are not sent every frame.

Our CD-quality audio example in Chapter 12 required isochronous packets with nine frames at 176 bytes and one frame at 180 bytes to maintain a 44.1-kHz sample rate with 16-bit samples. With 10 BTF of isochronous transaction protocol overhead per frame, this gives us a maximum of 190 BTF. Still a small load for USB. We have enough bandwidth for (1300 / 190)—almost seven separate CD-quality stereo audio channels on USB!

SIZING VIDEO DATA

This chapter deals with video, and one of the first things you realize about video data is that there is a lot of it. An NTSC TV screen, for example, is 720 x 480 x 24 bit = 1.0368 MBytes of data (a PAL TV screen is 720 x 576 x 24 bit = 1.244 MBytes of data). A TV picture is updated at 59.96 fields per second (two fields per frame), which results in a raw data rate of 31 MBps or 20,750 BTF. Because this **far** exceeds the maximum BTF of USB, it is easy to conclude that we cannot support full-screen, full-frame rate on USB. But let's see what some signal processing can do!

There are some easy choices we could make to reduce the data rate:

- Take fewer samples. A video signal is characterized by its YUV components (Y = luminance or brightness of the image, UV = chrominance or color depth of the image), and the 24 bits assumes full data capture (YUV = 4:4:4). The human vision system has poor color acuity, so it is possible to subsample the U and V components without the viewer noticing. By sampling both chrominance components at half the rate horizontally (YUV = 4:2:2), the sample size is reduced to 16 bits. By also sampling at half the rate vertically (YUV = 4:2:0), the sample size is reduced to 12 bits.
- Reduce the frame size. Industry standard sizes include CIF (352x288), QCIF (176x144) , and "small" (160x120). Figure 1-1 shows the resulting frame rate if we limit the maximum BTF for the video source to 1000.
- Reduce the frame rate. Flicker will be noticeable at frame rates below 15 fps and slow-motion effects will be evident at frame rates below 2 fps.

CIF	352 x 288 x 2 = 203 KB/frame	0.5 fps
QCIF	176 x 144 x 2 = 51 KB/frame	2 fps
Small	160 x 120 x 2 = 38.4 KB/frame	2.6 fps

Figure 1-1. Resulting frame rates at 1000 BTF

This scheme of transmitting raw video on USB does not produce very good results, and I should mention that a video camera operating at 1000 BTF would not be considered a good citizen. Names such as "data hog" come to mind. A single I/O device using this much of the USB bandwidth should have a WARNING label on the front because it will impact the operation of all other USB devices. If you must use an I/O device in this way, an acceptable solution is to install another USB host controller in the PC host. Each host controller can support 12 Mbps, so installing a second host controller would effectively produce 24 Mbps. Early USB video cameras operated this way and prompted many users to add another USB host controller just for the video camera.

Video Compression Is Essential

A better solution for reducing the data rate from the video source is to **compress** the video data, transmit the compressed data on USB, and then decompress the data on the PC host for display. A modern PC host has ample processor performance to decompress all styles of video encoding in software, so no cost is incurred at the host. Today's second-generation USB cameras use codecs and compression techniques that make them very usable in most PC applications.

Two main methods are used for video compression:

- **Intraframe**, also called spatial compression, in which the data within each frame is compressed individually, with no reference to other frames. Examples of codecs that mainly use intraframe compression are run-length encoding (RLE) and JPEG compression.
- **Interframe**, also called temporal compression, in which the compression concentrates on just the parts of the image that change from one frame to the next. Sometimes key frames are included to reset image data once the changes become excessive. Examples of codecs that mainly use interframe compression are frame-differencing and MPEG.

It is possible to combine these two methods in the same codec.

Figure 13-2 shows a typical second-generation camera. Some signal processing, such as the compression algorithm, is implemented in hardware at the camera, but much signal processing, such as white balance, color correction, dead-pixel correction, and lens correction, are implemented in software on the PC host. Sharing the data processing allows the hardware cost to be reduced while still delivering an excellent product.

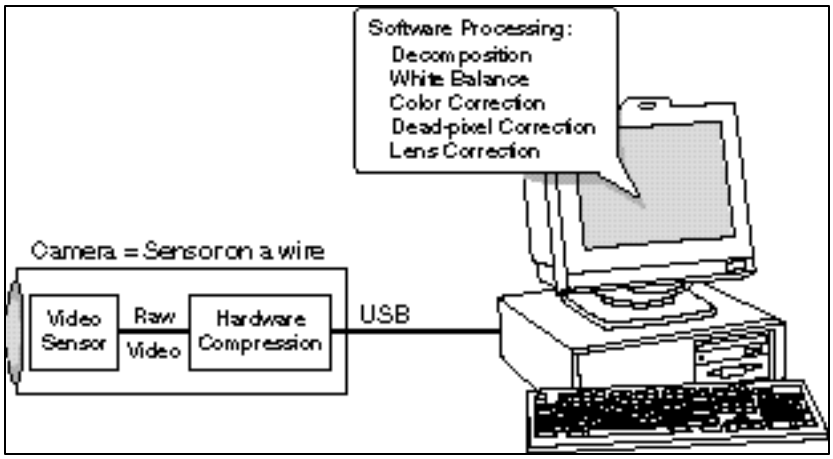
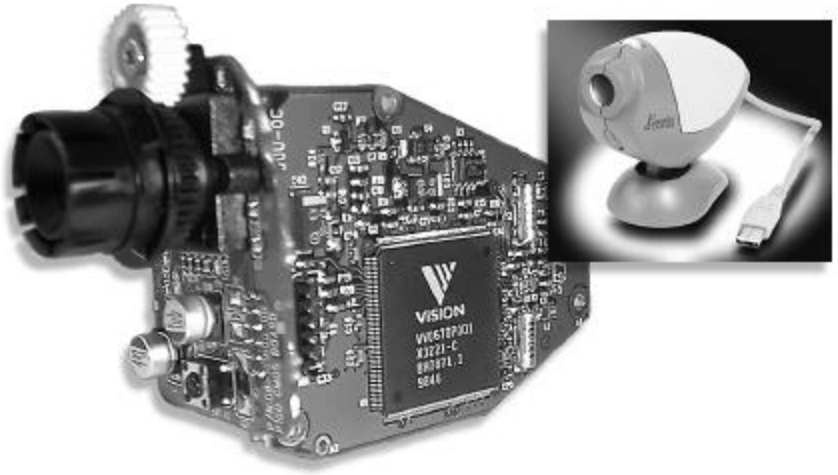


Figure 13-2. Digital signal processing is shared

Example 1: Videoconferencing Camera

Figure 13-3 shows the end-user view, and an OEM view, of a typical videoconferencing camera; this one is from Ezonics. The Ezonics product is a turnkey videoconferencing solution, and, because of its high functionality and low cost, the camera will be employed in multiple other applications. Inside the case is a robust video design from ST Microelectronics; Figure 13-4 shows a block diagram of this OEM solution.



Courtesy of ST Microelectronics (OEM design) and Ezonics Corp. (consumer design).

Figure 13-3. Video conferencing design

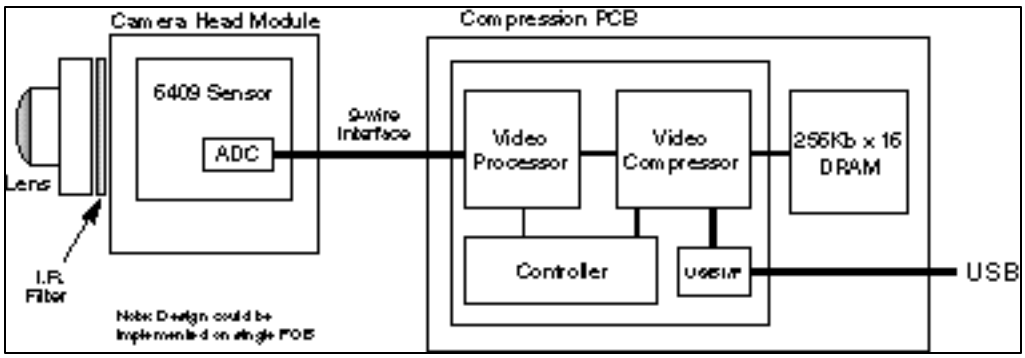


Figure 13-4. ST Microelectronics OEM reference design

The CMOS image sensor in the ST Microelectronics reference design is a single-chip device that includes the imaging array and all the control logic required to generate digital video. The block diagram (Figure 13-5) includes the photodiode array that is controlled by shift registers. The array generates pixel data that is digitized and formatted into two nibbles per pixel. The sensor also contains control functions for black-level calibration, exposure, image format, and a serial interface to allow control of the sensor from a coprocessor chip.

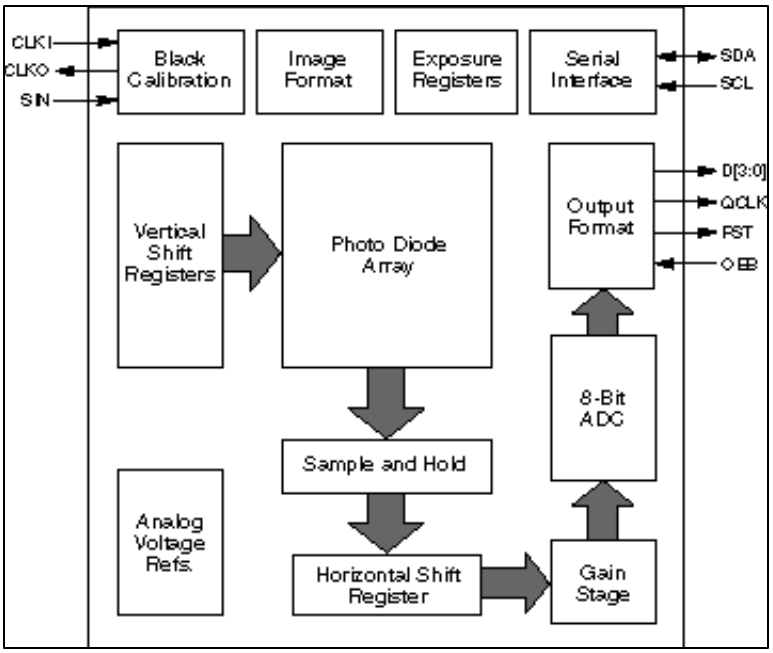


Figure 13-5. CMOS sensor block diagram

This CMOS sensor is controlled by a Color Coprocessor via an I2C bus. Nibble data from the sensor is strobed into the coprocessor for digital signal processing. The only other component required is a 4-Mbit DRAM, which provides the frame store and run-length buffer. The color coprocessor performs the color processing, implements a proprietary interframe compression encoder, and manages the USB interface. The whole design takes less than 100 mA, so it can be powered directly from the USB cable.

The reference design camera defines a single device descriptor, a single configuration descriptor, and four interface descriptors paired with four isochronous endpoint descriptors. The Default interface requests no USB bus bandwidth as required by the USB specification—this means that the device will

be successfully enumerated but will not use bus bandwidth until an application program selects an alternate interface. The alternate interfaces request more and more of the USB bandwidth.

The highest selection is equivalent to 968 BTF, and this allows a CIF format video to be displayed at 25 fps (depending on the amount of movement in the scene and performance of the PC host). If this bandwidth cannot be supported, then an alternate setting 2 requests 712 BTF. A third alternate setting requests 456 BTF, and if this low setting cannot be supported, then the user is requested to remove an existing isochronous device from the USB bus so the new camera can be used.

The accompanying device driver decodes the data stream into an RGB24 format for DirectShow.

Example 2: Composite Video

My second example accepts a standard composite video signal (S-Video, NTSC, PAL, or SECAM) and two audio channels, digitizes them, compresses them, and transmits them over USB using only four major components. The reference design for the Nogatech NT1004 is a small video dongle as shown in Figure 13-6. Also shown is a commercially available product from Belkin, the Videobus II, that implements the Nogatech reference design and adds comprehensive digital image processing software to enable creative users to be immediately productive.

Courtesy of Belkin (consumer design, left) and Nogatech, Inc. (OEM design, right).

Figure 13-6. Inputting digital audio and video data via USB

Figure 13-7 shows a block diagram of the Nogatech Reference Design. The heart of the design is a single-chip hardware audio/video compressor with integrated USB interface. A 4-Mb or 16Mb DRAM is required for frame storage and to buffer the isochronous USB data. A Philips SAA7111A is used to convert the composite video into digitally encoded video for the NT1004. An I^2C interface and HID driver are also included to manage the video source, if required, and for system setup.

Figure 1-7. Block diagram of Nogatech's reference design

In standard operation the NT1004 will provide the following services:

Compressed Video Channel: The NT1004 connects to YUV video digital source, scales the image on the fly horizontally and vertically, compresses the data, and sends it to the host computer via the USB port. The bandwidth usage can be set from 0.5 Mbps to 8 Mbps in 0.5-Mbps increments. The NT1004 scaler supports zoom-in–like effects by applying combinations of zooming and cropping built-in functions. The unique method of compression is a special design of Nogatech, to allow easy and fast software-only decompression. The decompression software driver will accept the compressed data and convert it back to standard video formats in less time than it takes to read raw video from any external port.

Video Source Control: Control and status monitoring is implemented with a built-in serial interface, or direct I/O pins. They support a direct-attach camera including a CCD sensor. These controls can also be used by the application

software to control and monitor other remote devices. In a videoconference application, for example, this allows the local or remote user to set the focus, zoom, and other parameters of the camera, or even to switch the camera to the power-down mode. The NT1004 also supports the use of an external capture button that can be mounted on the camera board and used for capturing still frames on the host computer disk. Still images are captured and sent via the USB in the best quality and resolution that the camera can provide.

Audio Source Control: The NT1004 includes an interface to a low-cost telephony audio codec. Stereo sound is interleaved with video isochronous data on the USB cable.

The accompanying WDM driver provides separate capture filters for the decompressed video data and the sound data. A TWAIN compliant driver for high resolution still capture is also provided.

Example 3: Digital Video Creation

My third example is a full-digital video creation and editing system from Dazzle Multimedia (Figure 13-8). Figure 13-9 shows a block diagram of this system. The central element is a hardware MPEG compressor from C-cube that Dazzle has integrated with an audio codec and a USB subsystem.

Courtesy of Dazzle Multimedia, Inc.

Figure 13-8. Digital video creation and editing system

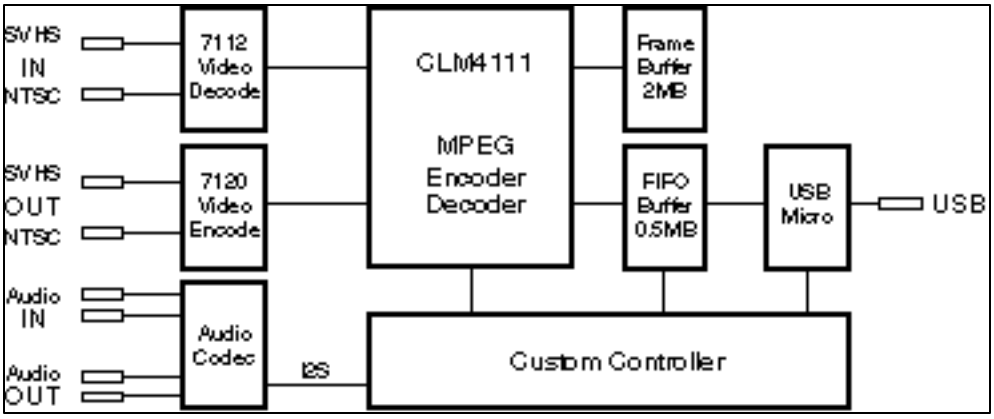


Figure 13-9. Block diagram of the hardware subsystem

MPEG, which stands for Moving Picture Experts Group, defines a set of standards used for coding audiovisual information (such as movies and music) in a digital compressed format.

The major advantage of MPEG compared to other video and audio coding formats is that MPEG files are much smaller for the same quality. This is because MPEG uses very sophisticated compression techniques. The MPEG strategy is to predict motion from frame to frame in the temporal direction and then to use discrete cosine transforms (DCTs) to organize the redundancy in the spatial directions. The DCTs are calculated on 8x8 blocks, and the motion prediction is done in the luminance channel on 16x16 blocks. In other words, given the 16x16 block in the current frame that is to be coded, a close match to that block is sought in a previous or future frame. The DCT coefficients of either the actual data, or the difference between this block and the close match, are quantized, which means that you divide them by some value to drop bits off the bottom end. We hope that many of the coefficients will then end up being zero. The results, which include the DCT coefficients, the motion vectors, and the quantization parameters, are Huffman-coded using fixed tables. This algorithm is well suited for a hardware encoding implementation, and a Pentium processor with MMX instructions can decode this MPEG data to produce a full-screen display in real time.

One side effect of the MPEG encoding is that data cannot be lost—the Pentium processor requires ALL of the encoded data to be able to decode the picture data. So, in contrast with the other digital video solutions presented in this chapter, this MPEG solution uses **bulk** transfers on USB because their delivery is guaranteed. During video capture the USB bus and the PC host's processor will be busy most of the time, and other activity is not recommended because this will result in some loss of video frames.

The USB hardware operates in two modes: It is either inputting digital video and audio into the PC host, or it is outputting digital video and audio from the PC host.

- Let's discuss the input function first. The analog video is digitized by a Philips SAA7112; this multistandard component can create YUV digital video from an NTSC, PAL, or SECAM signal source. This video is compressed at a real-time frame rate by the C-cube CLM4111 MPEG compressor. The compressor needs 2 MB of memory for a frame buffer and for run-length encoding. Compressed output is fed into a 0.5-MB FIFO that is emptied by a USB microcontroller using bulk transfers. At the same time, two audio channels are digitized using a codec, and this data is also sent to the PC host using bulk transfers.
- For playback the PC host will use bulk transfers to move video and audio data to the Dazzle hardware subsystem. The custom controller feeds the video data back through the CLM4111 MPEG encoder-decoder, which uses a Philips SAA7121 video encoder to create NTSC (or PAL/SECAM) video OUT. The custom controller uses an external audio codec to regenerate the sound.

The accompanying WDM driver provides filters for video and audio capture and for video and audio rendering. The Dazzle consumer product also comes with extensive applications software that enables the captured video to be edited and played back to a TV monitor or video recorder.

WINDOWS VIDEO SOFTWARE – DIRECTSHOW

DirectShow is one of the components of Microsoft's DirectX suite and it is the preferred method of playing video on a Windows platform. Microsoft defined DirectX to mask the rapid advances in PC platform hardware from the multimedia software developer. They defined a rich set of API's (Application Programmer Interfaces, such as DirectDraw, DirectMusic, DirectInput, DirectSound, etc.) and a robust set of device driver interfaces for hardware developers. This enables the hardware to advance and the software applications to improve at independent rates. In our example, it allows us to use any video conferencing camera that has a DirectX video capture minidriver. Figure 13-10 shows an overview of the DirectShow architecture.

Figure 13-10. The DirectShow Architecture is layered.

Note that the video data processing is done at the kernel level since it is time critical in nature. By enveloping all of the video processing in the stream class driver the amount of data copying is reduced and higher performance is possible. The operation of the stream class driver is defined by a user-mode component called a filter graph. Included on the CDROM, in the tools directory, is a Microsoft tool called GraphEdit which we shall use to construct and run some example filter graphs later in this section. I would also recommend that you download the DirectX Media DDK from www.microsoft.com/hwdev (it's free!).

A filter graph is composed of a collection of filters. Most filters can be categorized as:

- A source filter – which takes data from some source, such as a camera, disk file or the Internet and introduces it into the filter graph.
- A transform filter – which processes the data in some way and then, passes different format data on.
- An effects filter – which processes the data in some way but keeps the same data type, and passes it on.
- A renderer filter – which renders the data to a display, disk file or to any location that accepts media input.

Each filter has one or more INPUT or OUTPUT PINS as shown in Figure 13-11.

Figure 13-11. Filters are connected via PINS.

A filter graph is built up by interconnecting PINS. Let work through a few examples using GraphEdit to get a flavor of what's going on here. I am using a variety of videoconferencing cameras in these examples so your display won't look exactly like mine.

DIRECTSHOW PROJECTS

I am assuming that you have installed the device drivers and have attached a USB camera to your PC host which is running Windows 98 Second Edition, Millennium or Windows 2000. Start the GraphEdit program and select "Insert Filters" from the Graph menu. The Filter Graph Editor will display a menu of currently registered filters as shown in Figure 13-12.

Figure 13-12. Starting a new filter graph.

Select your USB camera from the Video Capture Sources list and then select "Video Renderer" from the DirectShow Filter list. Click and drag the Preview Pin (or ~Capture if there is no preview pin) of the Camera Source Filter to the Input Pin of the Video Renderer. The filter graph editor will check that these two pins are compatible and, if not, will insert a transform filter into the diagram. My first camera, for example, adds a Color Space Converter filter. Now click the PLAY button and an ActiveMovie Window will open and display what your video camera is pointing at. Your screen will look similar to Figure 13-13.

Figure 13-13. Displaying live video.

Depending upon the capabilities that your USB camera supplier included in their device driver, you may be able to alter the size of the Window or adjust contrast, brightness and color. Stop the video display and right click on the USB Camera filter box and choose Properties from the drop-down menu. A properties box will open and give you an opportunity to reconfigure your video capture filter.

I have supplied an Effects Filter (USBDBEEF.AX) on the companion CDROM that you should now copy to your hard drive and register. With an Explorer window open and usbdbeef.ax selected, choose RUN and enter "regsvr32.exe usbdbeef.ax". This will register the filter such that will now appear, as USBDBE Effects Filter, in the DirectShow filters menu. Choose Add Filters and include this Effects Filter in your Graph Editor workspace.

I created my Effects Filter by extending the Microsoft-supplied Effects Filter to include a "Motion Detect" option. The source code for this filter is included on the CDROM but you will need the DirectX Media SDK if you wish to edit and rebuild it. The motion detect filter compares the current frame with the previous frame to detect motion – small differences are ignored but large frame-to-frame differences cause the motion detect filter to start outputting frames. We will record these to disk later in this example to create a "motion activated VCR" but, for now, we will just view them on the screen. Edit your filter graph to resemble Figure 13-14 and click play.

Figure 13-14. Detecting movement with a filter.

You should now have two DirectShow windows on your screen, one live and one displaying the captured frames that are different. Hold still for a while then wave at the camera, then hold still again, then stop the recording. Right click on the live Video Renderer and select properties. Observe the total number of frames played. Now right click on the motion detector Video Renderer and select properties. Observe the much smaller number of frames captured; duplicate frames have been discarded. Since we are not recording all of the frames then we need some method of identifying the frames that are kept. I did this with another selection option in the USB Design By Example Effects Filter which copies the current date and time onto the viewable area of the frame. Install another copy of USBDBE Effects Filter into your filter graph and choose "Time Stamp" from the Property Page menu. When you click PLAY now it is easy to see the freezing effect of the motion detect filter.

The final step is to replace the motion detect video renderer with a file writer filter. We will save the file in AVI format so will need to include a filter for this data conversion. I also removed the preview renderer so that the operation of the program is silent i.e. an observer does not know that detected motion is being recorded. The final filter graph is shown in Figure 13-15 and is saved on the CDROM as MotionDetect.GRF.

Figure 13-15. The Motion Detector Filter Graph

Now place the USB camera in a place that you would like to monitor – such as your office, your yard or home. If anything is stolen it will now be very easy to identify the time and people involved since all of the motion will be captured on video (the PC host should, of course, be well hidden; you wouldn't want the criminals to steal the evidence!)

The captured video file can be played back using the standard Windows Media Player. I capture at least one frame every minute so, with no large detected movements, a day will play back in 48 seconds. The threshold for ignoring small movements is currently set in the program but, hopefully by the time you are reading this, it will be a property page setting.

Depending upon the performance of your PC host you could extend this application to cover multiple cameras. I would use a MIXER filter to combine the outputs of, say, up to 4 cameras, and feed this into the motion detector filter. The USB cameras should be set to a low capture rate so that the USB bandwidth is preserved.

A WebCam is a small extension of this program. A web page usually contains images that have been uploaded to the webserver and these images are displayed using an HTML tag:

<IMG SRC=”myhome.jpg” WIDTH=352 HEIGHT=288 BORDER=5>

The web page should have an additional tag in it’s <HEAD> which will refresh your web page every, say, 60 seconds:

<META HTTP-EQUIV=”refresh” CONTENT=”60”>

The technique, then, is to copy images from the PC Host that is collecting the images to the server that is hosting your web page. I have added a JPEG conversion filter and a remote file renderer to the filter graph in Figure 13-16. I also assume that you have a permanent connection to the web such as a cable modem or a DSL link. This filter graph will successfully upload an image to the web server every minute or whenever motion is detected. Several frames will not be displayed if there is excessive movement, but these are being stored locally for later review.

You’re done! You can now remotely monitor your home, office or yard using your web site.

Figure 13-16. A WebCam Filter Graph

USB-ENABLED VIDEO APPLICATIONS

Digital Microscope

Figure 13-17 shows a variation on the videoconferencing camera. Intel and Mattel are collaborating on the design of a digital microscope that will be used in schools and will be very popular in the home.

Rather than each student having to peer down an eyepiece to observe a specimen, a USB camera displays an image on the PC screen so that the whole class can see. The image is "live," so that focusing and positional adjustments can be made, and then a higher-resolution image can be captured and stored on disk. These images can be included later in a school report. Science was never this much fun when I went to school!

At home, children will be able to explore their world and then have creative fun doing things with their discoveries that they can show to their friends.

Courtesy of Intel Corp. and Mattel, Inc.

Figure 13-17. A digital microscope from Mattel-Intel

Biometrics

The decreasing cost of capturing complex data types such as images is bringing biometrics applications to the USB-enabled PC. Biometrics is a technology that verifies a person's identity by measuring a unique-to-the-individual biological trait. Biometric technologies include retinal/iris scanning, face-shape recognition, dynamic signature verification, voice recognition, and fingerprint identification.

Biometric identification is superior to lower-technology identification methods in common use today. Today, physical objects or behaviors-based-on-memory are used to identify a user. Physical objects include smart cards or magnetic-stripe cards; behaviors-based-on-memory include the act of entering a PIN number or a secret password.

The primary use of a physical object or behaviors-based-on-memory has a clear set of problems and limitations. Objects are often lost or stolen, and a behavior-based-on-memory is easily forgotten. Also, the use of a valid password on a computer network does not mean that an identity is genuine. These limitations decrease trust and increase the possibility of fraud. They are at the root of widespread distrust of the Internet and are the biggest weakness in true network security. Trust is the problem—without biometrics it is simply not possible to trust or prove that a person is really who they claim to be. Even worse—without biometrics the risk of fraud is very high, while paradoxically the **proving** of fraud is impossible.

Biometrics is a better alternative because it increases trust by creating a context of confidence and undeniable personal responsibility. It is very difficult to share your face or fingerprint. It is nearly impossible to replicate a voice pattern or duplicate the creation of a handwritten signature.

In the past—because of a lack of desktop computer speed and narrow available network bandwidth—there was no practical alternative to the use of physical objects and behaviors-based-on-memory. These old technologies flourished because there was no attainable practical alternative. The future destiny of computerized network security and identification is biometrics. The limiting factors of speed and bandwidth are now a thing of the past. The sensors and electronics to implement biometrics used to be what you saw in science fiction movies—now you can buy them for less than a hundred dollars.

Two interesting biometrics applications are the U.are.U fingerprint analyzer shown in Figure 13-18 and the IriScan application shown in Figure 13-19.

Fingerprint Identification

Image sensors, as described earlier in this chapter, are only the first step in capturing and verifying a fingerprint. The fingerprint sensor is often dirty because of its frequent use, and patterns in the shape of previous fingerprints are already on the sensor before we start. When a new finger is applied, the detector must create a new image knowing the previous dirt on the sensor.

The sensor could send the raw, new image up to the PC host for processing, but this is a security risk! It is easy to observe all of the data transactions on a USB cable, so a clever spy could record the data stream of a valid user's fingerprint and inject this data stream later. To prevent this type of fraud, the U.are.U sensor uses a high-performance USB microcontroller inside the unit to preprocess the data (Figure 13-18). It then manages a secure handshake with PC host-based software and sends encrypted results on the USB cable. Recording the data stream for later misuse will not work!

Courtesy of DigitalPersona, Inc.

Figure 13-18. Fingerprint identification on the PC

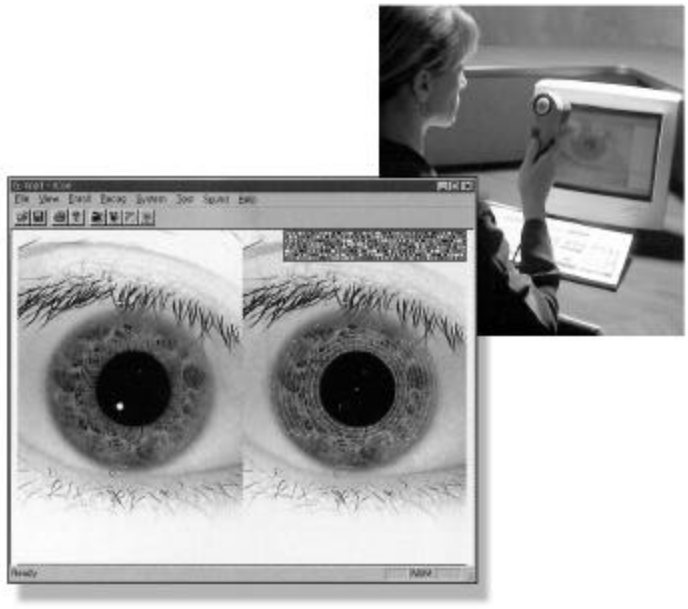
IriScan Application

IriScan's iris recognition technology identifies people by the unique patterns in the iris of the human eye. This capability is possible because no two irises are alike in their structural detail, not even between identical twins or triplets.

A high-definition (640x480x24-bit) image is required for effective iris pattern recognition (Figure 13-19). The intricate and personally distinct patterns of the iris are shaped from a complex information-rich meshwork of tissue. From this special pattern, a unique "bar code" is mathematically computed that is unlike any other in the population. In fact, the identification accuracy of IriScan's iris recognition technology outperforms DNA testing. This pattern, a very personal ID code, is safely and noninvasively acquired by a camera, analyzed, and encoded into a 512-byte IrisCode record. The IrisCode record is used to confirm a person's identity.

The sensor is able to detect eye movement in a live picture, so attempted fraud by using a photograph of a valid user will not work. Sorry! As the world's criminals get more sophisticated, then so must our protection schemes.

In the product configuration shown in Figure 13-19, processing is done on a PC host, requiring a secured cable connection from the imager to the PC host.



Courtesy of IriScan, Inc.

Figure 13-19. IriScan application on the PC

CHAPTER SUMMARY

Video capture with a PC host used to be expensive and difficult. USB has removed almost all of the complexity and cost from this solution. It is now easy to include video in your custom I/O application. Several manufacturers build custom USB controllers that are specifically designed to compress video so it can be transported efficiently over a USB cable. Some vendors use a proprietary compression algorithm and supply a Windows video capture driver to decompress the digital video. One vendor uses a standard MPEG encoding scheme to implement a combined video and audio solution.

The creation of video-based applications on a Windows platform has also been simplified with Microsoft's release of DirectX. The DirectShow component is easy to use with most of the hard work being handled by specialist programs at the kernel layer. Applications programs configure and manage Filter Graphs to control and direct video and audio stream actions.

The hardware proliferation and improved tools will encourage software developers to write more creative applications, which will mean more hardware vendors will make even better-quality video devices at lower cost. And this spiral will continue with the consumer gaining the most benefit.